Major Differences between C and C++

- C follows the procedural programming paradigm while C++ is a multi-paradigm language (procedural as well as object oriented).
- In C, the data isn't secured while data is secured (hidden) in C++ because of concept like data-hiding.
- C is a low-level language while C++ is a middle-level language.
- C uses the top-down approach while C++ uses the bottom-up approach.
- C is function-driven while C++ is object-driven.
- C++ supports function overloading while C does not.
- We can use functions inside structures in C++ but not in C.
- The NAMESPACE feature in C++ is absent in case of C.
- The standard input & output functions differ in the two languages. C uses scanf & printf while C++ uses cin>> & cout<< as their respective input & output functions.
- C++ allows the use of reference variables while C does not.
- C++ supports Exception Handling while C does not.

1. How do structures in C and C++ differ?

OR

What is a class? How a class is different from structure?

- The definition of the structure in C is limited to within the module and cannot be initialized outside its scope. Where as in C++ we can initialize the objects anywhere within the boundaries of the project.
- C++ structure is "class" which contains all functions of c and additional features as data binding and data hiding where in C it can have only members with user-defined data-types.
- By default C structure are Public while C++ structure are private.

2. An unsigned int can be as large as the signed int. Explain how?

- In signed, the sign needs one bit, and the rest are used for the integer. Unsigned uses this bit also as part of the integer. And as one extra bit available is equivalent to doubling the capacity, the unsigned integer has a maximum limit double that of a signed integer.
- Let us assume that the machine is 32bit wide. For a signed integer, when 32bits are allocated, 1 bit has to be used for storing the sign information. So the actual number can only be 31bit wide.
- For an unsigned integer, the whole 32bits can be used.
- So the range of signed int will be -(2^31) to (2^31)-1 and the range of unsigned int will be 0 to (2^32)-1.

3. What is early binding/ Static binding/ Compile-time polymorphism?

- **Early binding** (also called static binding) means the compiler is able to directly associate the identifier name (such as a function or variable name) with a machine address.
- Remember that all functions have a unique machine address. So when the compiler encounters a function call, it replaces the function call with a machine language instruction that tells the CPU to jump to the address of the function.

4. What are visibility modes/ specifiers/labels?

- Visibility mode is the keyword that controls the visibility and availability of inherited base class members in the derived class. It can be either private or protected or public.
- Private member isn't accessible outside the class or by any other derived class.
- Public members are accessible throughout the program.
- Protected members are accessible in the next class only.

5. What is constructor with default argument?

- We can also use default parameters in an object's constructor.
- A default argument is a value given in the declaration that the compiler automatically inserts if we don't provide a value in the function call. Only trailing arguments can be default arguments.
- For example, the cube class shown here maintains the integer dimensions of a cube. Its constructor defaults all dimensions to zero if no other arguments are supplied, as shown here:

```
#include <iostream>
class cube {
        int x, y, z;
        public:
        cube(int i=0, int j=0, int k=0) { x=i; y=j; z=k; }
        int volume() { return x*y*z; }
};
int main()
{
cube a(2,3,4), b;
cout << a.volume() << endl;
cout << b.volume();
return 0;
}
```

6. State operators which a friend function cannot overload.

A friend function cannot overload	
=	Assignment Operator
( )	Function call operator
[ ]	Subscripting operator
->	Class member access operator

7. What is inline function? When it is used?

- An inline function is a function that is expanded in line when it is invoked. That is, the compiler replaces the function call with the corresponding function code.
- A function takes a lot of extra time in executing a series of instructions for jumping, saving registers, pushing arguments into stack, etc. For small functions this steps will be overhead.
- Hence, Inline functions are best used for small functions such as accessing private data members. The main purpose of these one- or two-line "accessor" functions is to return state information about objects; short functions are sensitive to the overhead of function calls. Longer functions spend proportionately less time in the calling/returning sequence and benefit less from inlining.

8. What is pure virtual function? How is it declared?

OR

When do we make a function pure virtual? How it is declared?

- The requirement of a base class is to provide some properties to derived class and to create a base pointer required for run-time polymorphism. Other than these, we don't want our class to perform any task. In such a situation, we make a ***do-nothing*** function which serves as a placeholder only. This function is known as **pure virtual function**.
- A pure virtual function can be declared as: virtual void display ( ) = 0;
- A class containing pure virtual functions cannot be used to declare any object of its own. Such classes are called **abstract base classes.**

9. Write a statement using seekg() to go by the number 50 in the file.

OR

Give the function to locate the get pointer at 50th byte in random access file.

- ifstream infile;
  infile.seekg(50);

10. How does a constant defined by a 'const' differ from a constant defined by the preprocessor statement #define?

- The constant defined with **#define** have no type information and hence are not used to create typed constants like with **const** keyword.
  e.g. const int size = 10;
- C++ requires a **const** to be initialized to 0 where as in C it doesn't. By default, it initializes it to 0.
- **#define** constants are global in nature whereas **const** keyword constants are local to the file where it is declared.

11. What is destructor? How it is written in C++?

- A destructor is used to destroy the objects that have been created by a constructor.
- A destructor is a member function whose name is same as the class name but it is preceded by a tilde. e.g . ~A( ) { }
- A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.

12. What is an object? What do mean by array of objects? How they are created?

- Object is a variable of user-defined type **class.** Objects are also known as class variables.
  E.g. class A { }; x, y, z; - Here x, y and z are class variables (objects).
- We can also create arrays of variables that are of type **class**. Such variables are called ***arrays of objects.***
- We can create array of objects using – class-name objectname [size];
  E.g. – Objects of class Person - Person manager[5], worker[10];

13. What is the function of showpoint flag?

- The **ios : : showpoint** flag is an argument of setf( ) which displays trailing decimal points and trailing zeroes.
- E.g . – cout.setf(ios : : showpoint);

14. How does a main( ) function in C++ differ from main ( ) in C?

- C doesn't specify return type for main( ) hence main( ) { ……} is valid whereas in C++, the **main( )** returns a type **int** to the operating system and hence must have a return statement for termination. E.g. int main( ) {…….return 0; }
- If the function doesn't have a return statement, most C++ compilers will generate an error.

15. List out the operators which cannot be overloaded.

Operators that cannot be overloaded	
Sizeof()	size of operator
.	Membership operator
.*	Pointer-to-member operator
: :	Scope Resolution operator
? :	Dot operator

16. What is the difference between ios : app and ios : ate file opening modes?

- Both the **ios : : app** and **ios : : ate** takes to the end of the file when it is opened.
- The difference between the two is that the **ios : : app** allows us to add data to the end of the file only, while **ios : : ate** mode permits us to add data or to modify the existing data anywhere in the file.

17. State the difference between seekg() and tellg() function in random access file.

- **seekg()** moves the get pointer (input) to a specified location whereas **tellg()** gives the current position of the get pointer.

18. State the difference between tellg() and tellp() function in random access file.

- **tellg()** gives the current position of the get pointer whereas **tellp()** gives the current position of the put pointer.

19. What is overridden function? Or What do you mean by method overriding?

- Redefining a base class function in the derived class to have our own implementation is referred as overriding.
- Overriding the function signature is the same as the base class unlike overloading.
- E.g: When a Base class A's method **set()** is also defined in the derived class and when derived class object calls the method instead of calling the base class method the derived class version of **set()** is called. This is known as Method Overriding. **Set()** in derived class is known as **Overridden function** or **overridden method**.

20. Can we use the same function name for a member function of a class and an outside function in the same program file?

- Yes, we can definitely use the same name for a member function and outside function because a member function is treated differently and called only using an object so it becomes unique for that class. Hence, a normal function which is not a member function will be called normally.

21. What is dynamic binding/ Run-time polymorphism/ Late-binding?

- In some programs, it is not possible to know which function will be called until runtime (when the program is run). This is known as **late binding** (or dynamic binding).
- In C++, one way to get late binding is to use function pointers. To review function pointers briefly, a function pointer is a type of pointer that points to a function instead of a variable.
- With late binding, the program has to read the address held in the pointer and then jump to that address. This involves one extra step, making it slightly slower.
- However, the advantage of late binding is that it is more flexible than early binding, because decisions about what function to call do not need to be made until run time.

22. What is the application of scope resolution operator (: :) in C++?

- It permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.
- It is used to access an item that is outside the current scope.
- It is used for distinguishing class members and defining class methods.
- A major application of the scope resolution operator is in the classes to identify the class to which a member function belongs.

23. What do you mean by dynamic initialization of a variable? Give an example.

- Dynamic initialization of a variable means initializing the variables at run-time.
- E.g : **float** average **=** sum/i;
- Here average is declared as a variable of type float and at the same time it is assigned the value of sum/i. This is dynamic initialization.

24. What is Stream? Explain input/output streams with example.

- A stream is a sequence of bytes. It acts as either a *source* from which the input data can be obtained or as a *destination* to which output data can be sent.
- The source stream that provides data to the program is called the *input stream.* It receives the data to input from the input device like keyboard, mouse, etc. In C++, it receives the data in the program using **cin >>.**
- The destination stream that receives output from the program is called the *output stream.* It outputs the data to output device like monitor, printer, etc. In C++, it sends the data to the screen using **cout <<**.

25. What is self-referential structure?

- A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type.
- For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type.
- In C++, a class is also a type of structure which can be used to create another class using inheritance which consists of the properties of the previous class.

26. What is a friend class?

- When all the member functions of one class are declared as friend functions of another class, such a class is known as **friend class.**
- E.g. Class Z

```
{
        friend class X;          // All member functions of X are friends of Z
}
```

27. What is protected modifier in class? How does it differ from private modifier?

OR

What is the significance of protected modifier in class?

- A protected member variable or function can be accessed in child classes which are called derived classes.
- Private members cannot be accessed from anywhere other than member functions and friend functions whereas protected members can be accessed in the next class but not outside the class.

28. What is an abstract class or do-nothing classes?

- An abstract class is one whose role is only meant to lay a foundation for other classes that would need a common behavior or similar characteristics. Therefore, an abstract class is used only as a base class for inheritance. A class is made abstract by declaring its methods as "pure virtual methods.
- In short, a class which consists of at least one “pure virtual function” is known as an **abstract class** or **do-nothing class**.

29. What is the difference between ‘delete a’ and ‘delete [ ] a’?

- **‘delete a’** will destroy the memory used by a single variable **a** for reuse **‘delete [ ]a’** will destroy and release the memory of the entire array pointed to by **a.**
- If we use **delete a** when **a** is actually an array-type, it will create a memory leak and will not properly delete all the memory hold by the entire array **a[ ]**.

30. What is function overloading?

- When we have a family of functions with one function name but with different argument lists, the concept is known as 'function overloading'.
- E.g.: **Class A**

```
{
        int add(int a, int b);
        int add(int a, int b, int c);
        float add(double a, double b);
}
```

- Hera all the three functions have same name add, but it is called according to the no. of arguments passed by the calling function.

31. Discuss significance of new and delete operator.

OR

Advantage of New and Delete over malloc(), calloc() and free.

- The **new** and **delete** are two memory management in C++ used for dynamic storage allocation and deallocation.
- **'new'** is used to create memory space for any data-type such as arrays, structures and classes.
  Syntax: ptr-var = **new** data-type[size];
- **'delete'** is used to free the memory allocated to an object by **new** operator.
- **Advantages over malloc(), calloc() and free()**
  - ✓ '**new**' and '**delete**' are important as compared to malloc() and calloc(), free because they are functions and cannot be used to construct and destroy objects like new and delete.
  - ✓ New throws an exception if there is insufficient memory whereas malloc returns NULL
  - ✓ '**new**' and '**delete**' can be overloaded unlike malloc and free.

32. What is reference to object?

- A *reference* is an alias or an alternative name for an object. All operations applied to a reference act on the object to which the reference refers. The address of a reference is the address of the aliased object.
- A reference type is defined by placing the reference modifier **&** after the type specifier.
- If a function needs to modify the actual value of an argument (object) or needs to return more than one value, the argument must be *passed by reference (in C++ pass entire object)*.

33. What do you mean by operator overloading? Give syntax for it.

- C++ can give special meanings to operators, when they are used with user-defined classes. This is called *operator overloading*.
- Operator overloading is the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.
- E.g : When minus(-) is used on the left side of an operand it becomes a negative no and when it is used between two no's it performs substraction.
- We can overload an operator using a friend function also.
- We create an operator function for operator overloading in C++ as follows:

```
return-type class-name : : operator op(argument-list)
{
        Function body  // task to do
}
```

34. What does this pointer points to?

- The **this pointer** is a hidden pointer inside every class member function that points to the class object the member function is working with or the object that invokes it.
- This unique pointer is automatically passed to a member function when it is called and acts as an implicit argument to all member functions.

35. The two major components of objects are _____ and _____

- The two major components of objects are:
  - **i.** Data members (Properties) : E.g color, weight, size, etc.
  - **ii.** Member functions (Actions): Actions to be performed on these data members.

36. List characteristics of OOP.

- Emphasis on data rather than procedure
- Programs are divided into entities known as **objects**.
- Data Structures are designed such that they characterize objects. (**Classes**)
- Functions that operate on data of an object are tied together in data structures. (**Encapsulation**)
- Data is hidden and cannot be accessed by external functions. (**Data Hiding**)
- Objects communicate with each other through functions. (**Message Passing**)
- New data and functions can be easily added whenever necessary. (**Inheritance**)
- Follows bottom up design in program design.
- The ability to exist in various forms. (**Polymorphism**)

37. List advantages of Binary file.

- Reading and writing data is much faster because it requires no translation and hence time is saved.
- We can use all the special symbols and characters because binary language supports all.
- It is feasible when datasets are too larger hence becomes compact in size because saving the data in binary format rather than any other form would occupy less memory.

38. Explain getline ( ) function.

- **Getline()** reads a whole line of text that ends with a newline character (Return Key). This function is invoked using the object **cin.**
- **Syntax:** cin.getline(line,size);

39. Explain width ( ) function. Give example.

- **Width()** is used to define the width of a field necessary for the output of an data item.
- We can invoke it using the **cout** object.
- **Syntax:** cout.width(w); - w specifies the no of columns.

  E.g. : **cout.width(5);**

  cout << 543 << 12 ; will be displayed as right-justified on the screen assuming 5 column width.

40. Explain open ( ) and close ( ) function of file.

- **Open()** can be used to open multiple files that use the same file stream object.
- **Syntax:** **file-stream-class** stream-object;

  stream-object.open("filename");
- **E.g.** **ofstream** outfile;

  outfile.open("ABC"); - opens a file ABC for reading.
- **close()** closes the file handle opened with open().

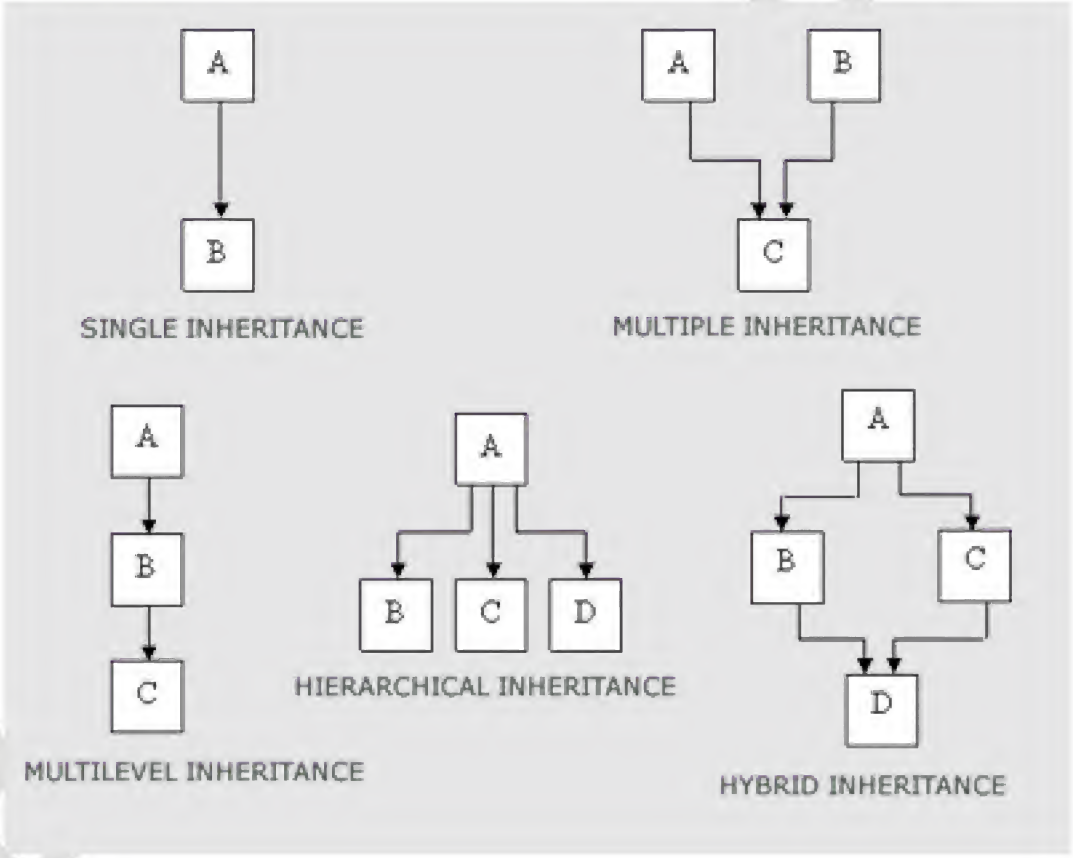
- E.g. outfile.close(); the last opened file here "ABC"

41. List types of Inheritance.

OR

Define Single, Multiple, Multi-level, Hierarchical and Hybrid Inheritance.

- **Single Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from one base class.
- **Multiple Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es)
- **Hierarchical Inheritance:** It is the inheritance hierarchy wherein multiple subclasses inherits from one base class.
- **Multilevel Inheritance:** It is the inheritance hierarchy wherein subclass acts as a base class for other classes.
- **Hybrid Inheritance:** The inheritance hierarchy that reflects any legal combination of other four types of inheritance.



42. Explain containership/ Aggregation/ Composition.

OR

Explain nested classes.

- **Containership:** When a class contains objects of other class types as its members, it is called containership or nesting. It is also called containment, composition, aggregation.
- The name of a nested class is local to its enclosing class.
- Member functions of a nested class follow regular access rules and have no special access privileges to members of their enclosing classes. Member functions of the enclosing class have no special access to members of a nested class.

E.g.:

```
Class Alpha { ……..};
Class Beta {……..};
Class Gamma
{
        Alpha a;
        Beta b;
        ……..
};
```

- Here all objects of **Gamma** will contains the objects **a** and **b.**

43. When do we make a class virtual?

- In multiple inheritance situations, when class A is inherited by class B and class C, and class D inherits from both B and C, then it inherits two instances of A which introduces ambiguity.
- To resolve this kind of ambiguity in inheritance relations, class can be made virtual by prefixing or suffixing **virtual** keyword before or after modifiers in inheritance.
- By declaring A to be virtual in both B and C, then D inherits directly from A, while B and C share the same instance if A.

44. Why is it necessary to overload an operator?

OR

Advantages/Importance of operator overloading

- A single operator or function can be used with different classes of objects or with normal operands, which can simplify a program's code.
- Overloading allows a user to control the behaviour of an operator with both user defined and standard classes.